LAPORAN PRAKTIKUM DATA STRUCTURE

HASH TABLES

Dosen Pengampu:

H. Fatchurrochman,M.Kom

Asisten Praktikum:

Fillah Anjany 230605110033

Oleh :

Muhammad Alif Mujaddid **240605110082**

Jurusan Teknik Informatika Fakultas Sains dan Teknologi

UIN Maulana Malik Ibrahim Malang

2025

A. PENDAHULUAN

1. Open Addressing Ide dari teknik ini adalah mencari alternative sel lain pada table ketika terjadi collision. Pada proses insertion, ketika indeks yang telah ditentukan dari Fungsi hash yang digunakan sudah berisi suatu item, maka akan mencari sel lain sesuai urutan menggunakan fungsi pencarian urutan. Beberapa startegi untuk menentukan fungsi pencarian urutan, yaitu:

a. Linier probing

b. Quadratic probing

c. Double hashing

Berikut ini listing program implementasi dari struktur data Hash Table dengan fungsi hash modulo key dengan table-size, dan menggunakan teknik linier probing. Pada linier probing, ketika terjadi collision maka akan mencari posisi terdekat dari posisi seharusnya pada table. Teknik ini hanya disarankan jika ukuran table dua kali lipat dari jumlah data.

```
public class Data {

    private int data;

    public Data(int data) {
        this.data = data;
    }

    public int getKey() {
        return data;
    }

}
```

```
public class HashTable {

    private Data[] hashArray;
    private int size;
```

```
        while (hashArray[hashVal] != null) {
            if (hashArray[hashVal].getKey()
                                    == key) {
                return hashArray[hashVal];
            }
            ++hashVal;
            hashVal %= size;
        }
        return null;
    }
} //akhir class HashTable
```

```
        while (hashArray[hashVal] != null) {
            if (hashArray[hashVal].getKey()
                                    == key) {
                return hashArray[hashVal];
            }
            ++hashVal;
            hashVal %= size;
        }
        return null;
    }
} //akhir class HashTable
```


```
        while (hashArray[hashVal] != null) {
            if (hashArray[hashVal].getKey()
                                    == key) {
                return hashArray[hashVal];
            }
            ++hashVal;
            hashVal %= size;
        }
        return null;
    }
} //akhir class HashTable
```

```
        while (hashArray[hashVal] != null) {
            if (hashArray[hashVal].getKey()
                                    == key) {
                return hashArray[hashVal];
            }
            ++hashVal;
            hashVal %= size;
        }
        return null;
    }
} //akhir class HashTable
```

```
public HashTable(int size) {
    this.size = size;
    hashArray = new Data[size];

}

public void displayTable() {
    System.out.println("Table: ");
    for (int j = 0; j < size; j++) {
        if (hashArray[j] != null) {
            System.out.println(" | "+j+"\t | "
                +hashArray[j].getKey() + " |");
        } else {
            System.out.println(" | "+j+"\t | -- |");
        }
    }
    System.out.println("");
}
```

```
public int hashFunc(int key) {
    return key % size;
}
```

```
public void insert(int data) {
    Data item= new Data(data);
    int key = item.getKey();
    int hashVal = hashFunc(key);
    while (hashArray[hashVal] != null) {
        ++hashVal;
        hashVal %= size;
    }
    hashArray[hashVal] = item;
}
```

```
public Data delete(int key) {
    int hashVal = hashFunc(key);
    while (hashArray[hashVal] != null) {
        if (hashArray[hashVal].getKey()
                                    == key) {
            Data temp = hashArray[hashVal];
            hashArray[hashVal] = null;
            return temp;
        }
        ++hashVal;
        hashVal %= size;
    }
    return null;
}
```

```
public Data find(int key) {
    int hashVal = hashFunc(key);
```

Lengkapi listing program tersebut dengan menanmbahkan class HashTableApp yang memiliki method main. Lakukan penambahan 10 item pada hash table berukuran 15, tampilkan isi data. Tulis output program yang telah anda lengkapi dan jelaskan bagaimana penentuan indeks (sel array) yang digunakan untuk menyimpan tiap item tersebut

```
public class HashTableApp {
    Run | Debug
    public static void main(String[] args) {
        HashTable hashTable = new HashTable(size:15);
        int[] dataArray = {12, 44, 13, 88, 23, 94, 11, 39, 20, 16};
        for (int data : dataArray) {
            hashTable.insert(data);
        }
        System.out.println(x:"=== Isi Hash Table Setelah Insert 10 Data ===");
        hashTable.displayTable();
        int keyToFind = 23;
        if (hashTable.find(keyToFind) != null) {
            System.out.println("Data dengan key " + keyToFind + " ditemukan di tabel.");
        } else {
            System.out.println("Data dengan key " + keyToFind + " tidak ditemukan.");
        }
    }
}
```

```
=== Isi Hash Table Setelah Insert 10 Data ===
| 0     | 88 |
| 1     | 16 |
| 2     | -- |
| 3     | -- |
| 4     | 94 |
| 5     | 20 |
| 6     | -- |
| 7     | -- |
| 8     | 23 |
| 9     | 39 |
| 10    | -- |
| 11    | 11 |
| 12    | 12 |
| 13    | 13 |
| 14    | 44 |

Data dengan key 23 ditemukan di tabel.
```

2. Tambahkan 5 item lain pada hash table tersebut. Tampilkan isi table sebelum penambahan (10 item) dan setelah penambahan (15 item). Dimana posisi 5 item yang baru saja ditambahkan? Jelaskan!

```
=== Isi Hash Table Setelah Insert 10 Data ===
| 0      | 88 |
| 1      | 16 |
| 2      | -- |
| 3      | -- |
| 4      | 94 |
| 5      | 20 |
| 6      | -- |
| 7      | -- |
| 8      | 23 |
| 9      | 39 |
| 10     | -- |
| 11     | 11 |
| 12     | 12 |
| 13     | 13 |
| 14     | 44 |

Data dengan key 23 ditemukan di tabel.
```

```
94          int[] dataBaru = { 17, 32, 45, 57, 68 };
95          for (int d : dataBaru) {
96              hashTable.insert(d);
97          }
98
99          System.out.println(x:"=== Isi Tabel Setelah Penambahan ===");
100         hashTable.displayTable();
```

```
=== Isi Tabel Setelah Penambahan ===
Table:
| 0      | 88 |
| 1      | 16 |
| 2      | 17 |
| 3      | 32 |
| 4      | 94 |
| 5      | 20 |
| 6      | 45 |
| 7      | 57 |
| 8      | 23 |
| 9      | 39 |
| 10     | 68 |
| 11     | 11 |
| 12     | 12 |
| 13     | 13 |
| 14     | 44 |
```

3. Pada method insert terdapat baris code while (hashArray[hashVal] != null) { ++hashVal; hashVal %= size; } Apa yang terjadi ketika bari tersebut dihapus dan program dijalankan untuk menambahkan item? Jelaskan!
Program tidak akan mengecek apakah slot sudah terisi atau belum.
Jika dua data memiliki hasil hash yang sama (collision), data baru akan langsung menimpa data lama di posisi yang sama.
Akibatnya, data lama akan hilang dan tidak bisa ditemukan kembali melalui operasi find() atau displayTable().

4. Lakukan pencarian berdasarkan key tertentu pada class HashTableApp!, jelaskan bagaimana proses pencarian pada method find?

```
Data dengan key 23 ditemukan di tabel.
```

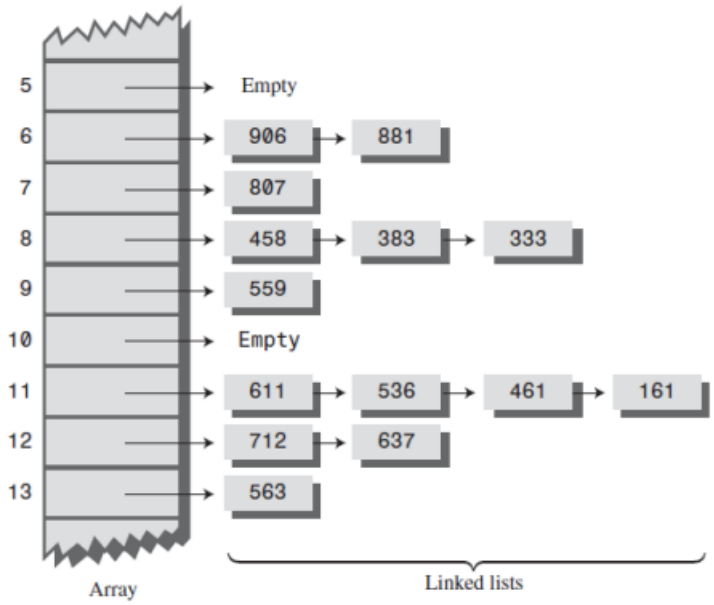
```
int keyToFind = 23;
if (hashTable.find(keyToFind) != null) {
    System.out.println("Data dengan key " + keyToFind + " ditemukan di tabel.");
} else {
    System.out.println("Data dengan key " + keyToFind + " tidak ditemukan.");
}
```

5. Lakukan hapus item berdasarkan key tertentu pada class HashTableApp!, jelaskan bagaimana proses pencarian pada method delete?

```
=== Isi Hash Table Setelah Insert 10 Data ===
Table:
| 0     | 88 |
| 1     | 16 |
| 2     | -- |
| 3     | -- |
| 4     | 94 |
| 5     | 20 |
| 6     | -- |
| 7     | -- |
| 8     | 23 |
| 9     | 39 |
| 10    | -- |
| 11    | 11 |
| 12    | 12 |
| 13    | 13 |
| 14    | 44 |

Data dengan key 23 ditemukan di tabel.
Table:
| 0     | 88 |
| 1     | 16 |
| 2     | -- |
| 3     | -- |
| 4     | 94 |
| 5     | 20 |
| 6     | -- |
| 7     | -- |
| 8     | -- |
| 9     | 39 |
| 10    | -- |
| 11    | 11 |
| 12    | 12 |
| 13    | 13 |
| 14    | 44 |
```

6. Separate Chaining / Open Hashing Pada teknik ini, permasalahan collision diselesaikan dengan menambah seluruh elemen yang memiliki nilai sama pada sebuah sel. Hal ini dilakukan dengan cara menyediakan sebuah linked list untuk setiap elemen yang memiliki nilai hash yang sama. Tiap sel pada hash table memiliki sebuah lisked list yang berisi data/elemen.



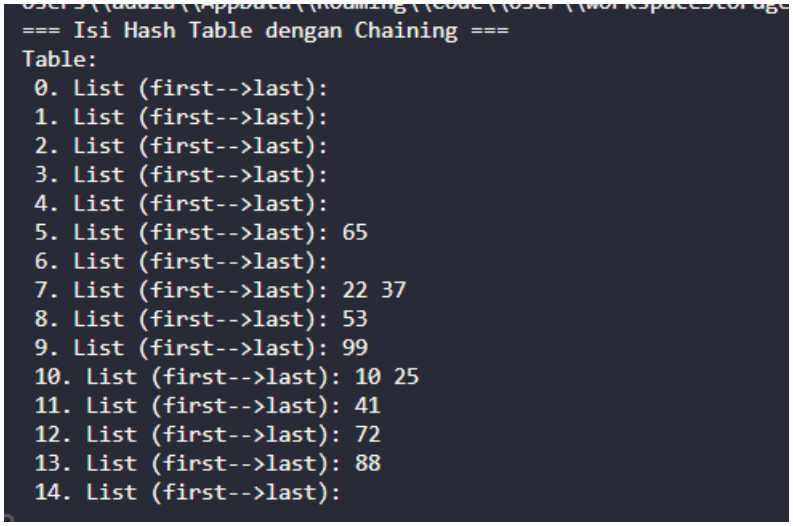
Gambar 8.1 Contoh Separate Chaining

Lengkapi listing program tersebut dengan menambahkan class HashChainApp yang berisi method main. lakukan penambahan data sebagaimana soal nomer 1. Jalankan program dan tulis outputnya. jelaskan bagaimana penentuan indeks (sel array) yang digunakan untuk menyimpan tiap item tersebut!

```
public class HashChainApp {
    Run | Debug
    public static void main(String[] args) {
        HashTable hashTable = new HashTable(size:15);

        int[] data = { 10, 25, 37, 22, 41, 53, 65, 72, 88, 99 };

        for (int d : data) {
            hashTable.insert(d);
        }
        System.out.println(x:"=== Isi Hash Table dengan Chaining ===");
        hashTable.displayTable();
    }
}
```


=== Isi Hash Table dengan Chaining ===
Table:
0. List (first-->last):
1. List (first-->last):
2. List (first-->last):
3. List (first-->last):
4. List (first-->last):
5. List (first-->last): 65
6. List (first-->last):
7. List (first-->last): 22 37
8. List (first-->last): 53
9. List (first-->last): 99
10. List (first-->last): 10 25
11. List (first-->last): 41
12. List (first-->last): 72
13. List (first-->last): 88
14. List (first-->last):